Contents

Configuring tracing using the Telemetry API

5 minute read   page test

Istio provides a Telemetry API that enables flexible configuration of tracing behavior. The Telemetry API offers control over tracing options such as sampling rates and custom tags for individual spans, as well as backend provider selection.

Before you begin

1. Ensure that your applications propagate tracing headers.
2. Follow the tracing installation guide located under Integrations to install your preferred tracing provider.

Telemetry API: Tracing Overview

The Telemetry API offers tracing behavior configuration control over the following at the mesh, namespace, and workload levels:

- **provider selection** - allows selection of backend providers for reporting.
- **sampling percentage** - allows control of the rate of trace sampling applied to received requests *for which no prior sampling decision has been made*.
- **custom tags** - allows control over any custom tags to add to each generated tracing span.
- **tracing participation** - allows opting services out of reporting spans to the selected tracing providers.

Workload Selection

Individual workloads within a namespace are selected via a `selector` which allows label-based selection of workloads.

It is not valid to have two different `Telemetry` resources select the same workload using `selector`. Likewise, it is not valid to have two distinct `Telemetry` resources in a namespace with no `selector` specified.

Scope, Inheritance, and Overrides

Telemetry API resources inherit configuration from parent resources in the Istio configuration hierarchy:

1. root configuration namespace (example: `istio-system`)
2. local namespace (namespace-scoped resource with **no** workload `selector`)
3. workload (namespace-scoped resource with a workload `selector`)

A Telemetry API resource in the root configuration namespace, typically `istio-system`, provides mesh-wide defaults for behavior. Any workload-specific selector in the root configuration namespace will be ignored/rejected. It is not valid to define multiple mesh-wide Telemetry API resources in the root configuration namespace.

Namespace-specific overrides for the mesh-wide configuration can be achieved by applying a new `Telemetry` resource in the desired namespace (without a workload selector). Any `Tracing` fields specified in the namespace configuration will completely override the field from the parent configuration (in the root configuration namespace).

Workload-specific overrides can be achieved by applying a new Telemetry resource in the desired namespace *with a workload selector*. Any `Tracing` fields specified in the namespace configuration will completely override the field from any parent configuration (root configuration or local namespace).

Using the Telemetry API for Tracing Configuration

Configuring tracing providers

Tracing providers are the backend collectors and processors that receive tracing spans and process them for storage and retrieval. Example providers include Zipkin, Jaeger, Lightstep, Datadog, and Apache SkyWalking.

For Istio, tracing providers are configured for use within the mesh via `MeshConfig`. To configure new providers to use in tracing, edit the `MeshConfig` for your mesh via:

```
$ kubectl -n istio-system edit configmap istio
```

The full set of configuration options is described in the reference docs for MeshConfig. Typical configuration includes service address and port for the provider, as well as establishing a limit on max tag length supported by the provider.

Each configured provider *must* be uniquely named. That name will be used to refer to the provider in the Telemetry API.

An example set of provider configuration in `MeshConfig` is:

```
data:
  mesh: |-
      extensionProviders: # The following content defines two example traci
ng providers.
      - name: "localtrace"
        zipkin:
          service: "zipkin.istio-system.svc.cluster.local"
          port: 9411
          maxTagLength: 56
      - name: "cloudtrace"
        stackdriver:
          maxTagLength: 256
      defaultProviders: # If a default provider is not specified, Telemetry
 resources must fully-specify a provider
          tracing: "cloudtrace"
```

Configuring mesh-wide tracing

behavior

Telemetry API resources inherit from the root configuration namespace for a mesh, typically `istio-system`. To configure mesh-wide behavior, add a new (or edit the existing) `Telemetry` resource in the root configuration namespace.

Here is an example configuration that uses the provider configuration from the prior section:

```yaml
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: mesh-default
  namespace: istio-system
spec:
  tracing:
  - providers: # only a single tracing provider is supported at this time
    - name: localtrace
    customTags:
      foo:
        literal:
          value: bar
    randomSamplingPercentage: 100
```

This configuration overrides the default provider from `MeshConfig`, setting the mesh default to be the `"localtrace"` provider. It also sets the mesh-wide sampling percentage to be `100`, and configures a tag to be added to all trace spans with a name of `foo` and a value of `bar`.

Configuring namespace-scoped tracing behavior

To tailor the tracing behavior for individual namespaces, add a `Telemetry` resource to the desired namespace. Any tracing fields specified in the namespace resource will completely override the inherited field configuration from the configuration hierarchy. For example:

```
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: namespace-override
  namespace: myapp
spec:
  tracing:
  - customTags:
      userId:
        header:
          name: userId
          defaultValue: unknown
```

When deployed with into a mesh with the prior mesh-wide example configuration, this will result in tracing behavior in the `myapp` namespace that sends trace spans to the `localtrace` provider and randomly selects requests for tracing at a `100%` rate, but that sets custom tags for each span with a name of `userId` and a value taken from the `userId` request header. Importantly, the `foo: bar` tag from the parent configuration will not be used in the `myapp` namespace. The custom tags behavior completely overrides the behavior configured in the `mesh-default.istio-system` resource.

> Any tracing configuration in a `Telemetry` resource completely overrides configuration of its parent resource in the configuration hierarchy. This includes provider selection.

Configuring workload-specific tracing behavior

To tailor the tracing behavior for individual workloads, add a `Telemetry` resource to the desired namespace and use a `selector`. Any tracing fields specified in the workload-specific resource will completely override the inherited field configuration from the configuration hierarchy.

For example:

```yaml
apiVersion: telemetry.istio.io/v1alpha1
kind: Telemetry
metadata:
  name: workload-override
  namespace: myapp
spec:
  selector:
    matchLabels:
      service.istio.io/canonical-name: frontend
  tracing:
  - disableSpanReporting: true
```

In this case, tracing will be disabled for the `frontend` workload in the `myapp` namespace. Istio will still forward the tracing headers, but no spans will be reported to the configured tracing provider.

It is not valid to have two `Telemetry` resources with workload selectors select the same workload. In those cases, Istio tracing behavior is undefined.